Advanced Graphics on Sun™
An Introduction to Texture Mapping

Technical White Paper

Sun microsystems

Please
Recycle

Contents

Introduction 1

Over the last several decades, the field of computer graphics has grown from specialized applications in niche markets such as mechanical design and mapping, to broad usage in areas ranging from electronic design and animation to graphics arts and multimedia.

Advances in graphics technology have enabled this explosion in applications. Sophisticated ASICs, for instance, have provided low-cost 2-D and 3-D graphics accelerators. Likewise, powerful software technology has led to the proliferation of high quality rendering tools, development tools, graphical user interfaces, and the emergence of standards in application programing interfaces (APIs).

As computers and media continue to converge, there is increasing demand for visual realism in a wide variety of applications. Texture mapping has emerged as an important tool for providing synthetic realism in applications from traditional design and simulation packages, to today's interactive "virtual reality" games.

This document serves as an introduction to texture mapping, its applications, techniques, and approaches. Texture mapping technology is described in Chapter 2. Chapter 3 discusses graphics architectures and their impact on texture mapping with particular focus on the Creator3D graphics architecture. Software interfaces which provide texture mapping support are covered in Chapter 4.

Texture Mapping and Visual Realism

Providing a sensation of realism is increasingly important to advanced graphics applications, whether they are used in military simulation or for consumer entertainment. The complexity of real-world scenes is necessarily great, and finding an effective means to meet the goals of realism while sustaining high image quality and/or real-time display rates is problematic.

Texture mapping (or pattern mapping) provides an extremely powerful technique for rendering visual detail, without the requirement for explicit modeling of the geometry of the textured object. Texture mapping lends a realistic appearance to a computer generated scene, particularly where the number or complexity of the rendered objects is significant.

The rendering of repetitive objects such as trees, windows, or bricks can be easily and quickly achieved with texture mapping. Drawing a large flat brick wall without texture mapping would require rendering geometric structure for each brick in the wall. By using texture mapping, one geometric object is drawn (the wall) and a 2-D image of a brick wall is "stuck" to the rectangle which represents the wall.

Applications for Texture Mapping

From scientific and engineering applications to highly artistic computer-assisted animation projects, texture mapping is finding utility in a wide range of applications.

Advanced Computer-Aided Design

Today's corporations have large investments in 3-D mechanical design data (CAD/CAM/CAE, and architectural), which must be used effectively throughout the organization.

It is common to subject these 3-D designs to finite element modeling, constructive solids geometry for interference checking, stress analysis, and a variety of other compute-intensive 3-D analysis techniques. Using 3-D texture mapping, volumetric analysis data can be mapped to 3-D design data enabling engineers to evaluate designs by their internal properties.

Simulation and design styling applications require sophisticated lighting, transparency, and antialiasing capabilities. The ability to map 2-D textures to surfaces enables design engineers to provide realistic representations of models which reduces the need to produce full-scale physical mock ups.

Geographic Information Systems (GIS)

GIS applications vary broadly in the problems they help solve and the techniques they use. For example, some GIS applications provide only simple 2-D mapping capabilities appropriate for land use analysis, utility management, emergency services, and other uses. Recent availability of digital terrain data and low-cost global positioning equipment has driven the use of 3-D data for common GIS display and analysis.

Representing geographically-oriented data using wireframe or solids modeling can provide an added dimension to data interpretation. The ability to visually represent terrain height enables users to more quickly understand the significance of their data as it relates to surrounding terrain (e.g., flood plains). Of increasing interest to these users is the ability to map textures in the form of satellite imagery to the 3-D surface data to more effectively simulate vegetation, geologic formations, and waterways.

Earth Sciences

Earth sciences applications, such as those used in the oil and gas industry, typically manipulate large volumes of both 2-D and 3-D seismic analysis and well-log data. Scientists interpreting this data strive to understand its significance in the context of local geological strata to determine the likelihood of the presence of mineral deposits, and the effort required to retrieve them.

In the case of 3-D seismic data, techniques which permit the visualization of underground strata often rely upon the generation of data 'slices' which provide the user with more powerful visualization capability. 3-D slices can be computed in a number of ways, with corresponding data complexity requirements and computational demands. Of growing interest to geoscientists is the ability to apply 3-D texture mapping techniques to the data slice in an accelerated fashion, allowing more flexible and rapid viewing.

Medical Imaging

Medical imaging applications have experienced substantial growth in recent years, both in terms of usage and sophistication. While medical imaging applications traditionally have been oriented toward the manipulation and display of 2-D data such as X-ray, computer-assisted tomography, and related areas, the use of 3-D data is growing rapidly.

The use of computer graphics in medical imaging has recently expanded to include 3-D vector graphics and solids modeling for diverse applications ranging from microsurgery and non-surgical exploration using micro-optics to facial reconstruction (both forensic and rehabilitative), and CT scans (volumetric rendering). The latter two examples in particular are prime candidates for the use of texture mapping.

CT scan information, much like seismic data, utilizes a large number of data slices. The flexibility to view and change the data’s internal structure rapidly is very important, and the pragmatic application of advanced texture mapping techniques can make this possible. Applications like facial reconstruction and plastic surgery already employ 3-D graphics capabilities, and with texture mapping these programs can achieve better quality, and more realistic representation.

Graphic Arts

Graphic arts applications perform the creation and manipulation of line art and images as used in a variety of areas including publishing, illustration, and advertising. Graphic arts applications focus on the production of photorealistic and synthetic images based upon computer generated geometry or bit-mapped textures. To create such effects, graphic arts applications increasingly are using advanced graphics functionality such as lighting and shading, transparency, image processing, and ray tracing. Manipulation of the data via computer modeling (versus real-world photography) can also give the artist greater freedom in introducing artificial objects into the scene, the effect of which can be very visually compelling.

Texture mapping techniques allow these users to increase the sophistication and quality of their final images by the application of textures to 3-D geometry in combination with other advanced graphics capabilities. For example, users can place wood grain on tables and patterns on shower curtains to create realism, or produce synthetic background textures yielding artistic effects.

Simulation, Animation, and Virtual Reality

Simulation and animation applications vary in their rendering requirements based upon the type of data (2-D, 3-D, lighted and shaded, antialiased, etc.) to be rendered and whether the geometry is structured for playback only or for interactive editing.

Movies such as Jurassic Park™ and Toy Story™ are testimonies to computer-aided animation. Commercial and military flight simulators, arcade-style computer games, and even virtual reality applications benefit from very high display rates of dynamically changing scenes.

Texture mapping allows computer-generated 3-D figures to appear more realistic. For example, a table can appear to be made of marble, wood, or glass. When realism is achieved, objects like a table, posses a sense of mass. Combat simulations and virtual reality environments which allow people to experience (and even control) their real-world equivalents are typically heavily dependent on the use of texturing techniques.

Visualization

Visualization technology represents the application of computer graphics technology to the visualization of medical or scientific data, traditionally represented in static media such as plots, charts, x-ray or hard-copy images. The application of visualization software algorithms, combined with graphics workstations, permits the data to become ‘dynamic’, allowing the engineer or scientist to manipulate and visualize the data in real-time.

Visualization applications vary in their requirements and are dependent upon whether they manage 2-D or 3-D data and image depth (8, 12 or 24-bit). Some visualization applications encompass image processing of static data, while others generate and manipulate 3-D models or simulations.

Visualizing the results of a thermodynamic simulation as a 3-D texture mapped solid gives engineers and scientists insight into internal structure or events.

Texture Mapping on Sun

Because texture mapping combines image processing techniques with the manipulation of three dimensional geometric objects, it requires a graphics architecture that excels at both disciplines.

Unfortunately, the image manipulation capabilities as well as the I/O architectures of many current graphics accelerators are limited by their emphasis on the computation of geometric data. The location of many such 3-D accelerators on slow peripheral buses creates greater overhead in communicating with the CPU and memory, causing imaging and texture mapping applications to suffer.

Creator3D Graphics Systems

Sun's Creator3D graphics architecture provides an innovative approach which accelerates both 3-D graphics and imaging applications without compromising performance or application flexibility. Sun™ Ultra™ workstations equipped with Creator3D graphics systems support 2-D and 3-D texture mapping applications written in a wide range of graphics APIs without adding arbitrary limitations to applications or extra cost to the base computing platform.

Texture Mapping Background 2

As previously mentioned, applications of texture mapping technology ranges from highly interactive applications where texture mapped objects are manipulated in real-time to applications where image quality is of primary importance.

Different texture mapping techniques require a range of calculations to be performed: as a result, selecting texture mapping methods often involves a trade off between interactivity and final image quality. Techniques used to provide convincing images of skin moving over a dinosaur's frame are likely inappropriate for a high-speed interactive computer game.

This section presents an overview of texture mapping terminology and techniques. While this document primarily focuses on interactive texture mapping applications, sections that follow will also highlight more advanced, computationally demanding techniques which are generally used for animation.

Basic Texture Mapping Techniques

Textures can be thought of as simple rectangular arrays of data (2-D textures) or volumes containing multiple rectangular arrays of data (3-D textures). The individual values in a 2-D texture are known as *texels* (texture elements) and are analogous to pixels on a computer screen. In a 3-D texture, these individual elements may be referred to as *voxels* to indicate their placement within a volume of image data.

The data contained in, or referenced by a texel (or voxel) can be color data, intensity or luminance data, or may combine color and alpha data (RGBA).

Texture mapping algorithms map rectangular texture data to non-rectangular regions. In addition to mapping a single texture, texture mapping algorithms must also be able to map multiple textures to a single object, and maintain the correct appearance no matter how the object is transformed (rotated, scaled, and projected). Lighting and perspective calculations must also be performed.

Given the wide range of operations that may be performed on texture mapped objects, filtering operations are required to produce the correct final pixel color. Discrete texels from the initial texture may end up being mapped to more than one pixel for final display to the screen. Alternately, one screen pixel may cover several texels, depending on the orientation and scaling of the texture mapped object.

2-D Texture Mapping

2-D texture mapping works by combining either synthetic or photographic images with traditional vertex-based geometric objects.

The easiest way to understand 2-D texture mapping is by imagining the process of placing a flexible decal on a two- or three-dimensional object. Another approach is to imagine "draping" a patterned cloth over a three dimensional object. In both cases, the pattern or texture assumes the shape of the underlying geometry.

Figure 2-1 illustrates a basic use of 2-D texture mapping. In this case, the floor below the wagon has been textured to represent a wood appearance. A second texture was used to apply the "Sun Flyer" decal to the wagon in the image. Other advanced graphics methods were employed to produce lighting, shading, and highlights.

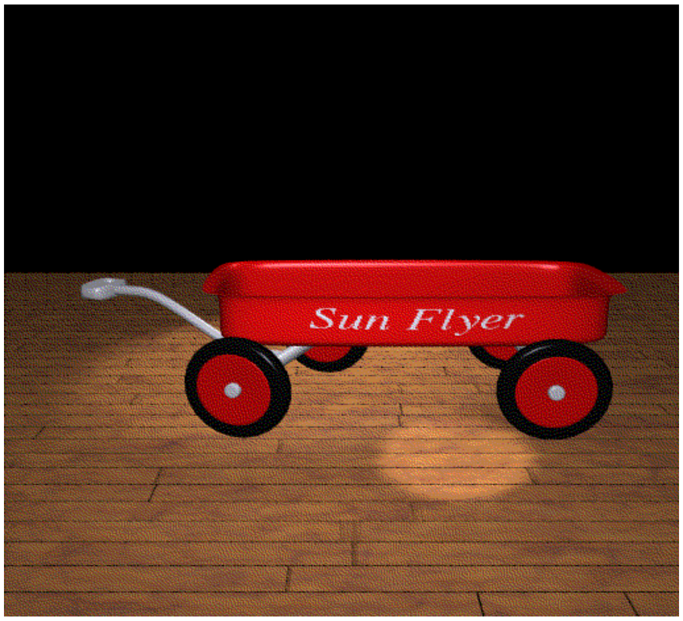

Figure 2-1 Texture mapping provides considerable flexibility in defining the appearance of objects and their scenes

3-D Texture Mapping

In contrast to 2-D texture mapping which maps a 2-D texture or image onto the surface of geometric data, 3-D texture mapping extends a 3-D texture *through* a three dimensional geometric object. Like 2-D textures, 3-D textures can be either captured (medical or seismic data) or programmatically generated (such as a computer-generated marble pattern).

3-D texture mapping approximates volume rendering but at a lower computational cost. As a result, 3-D texture mapping applications are able to provide interactive performance on a wider range of hardware platforms.

2-D texture mapping is used principally to enhance 3-D data. In contrast, the texture in 3-D texture mapping generally is the data of interest. 3-D texture mapped objects can be cross-sectioned or sliced to reveal otherwise unseen internal details.

Figure 2-2 illustrates an arbitrary slicing plane taken through three dimensional data from a CT Scan. Data is extracted from the 3-D texture and a 2-D texture is then mapped to the intersecting plane. This approach differs from traditional volume rendering where actual voxel values are probed with rays projected from the user's viewpoint and tiny triangles representing the voxels are then lit and shaded.

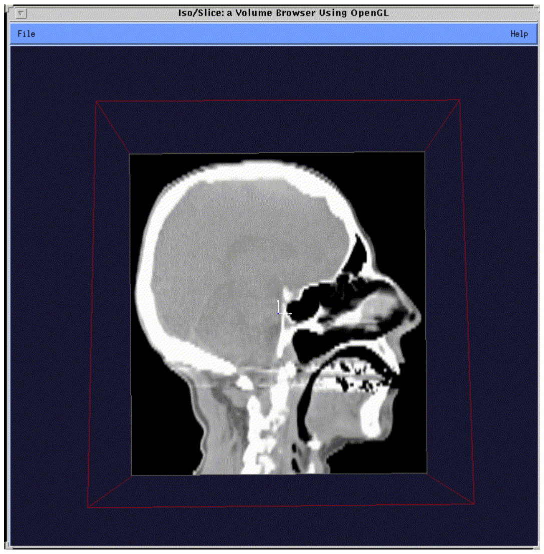

Figure 2-2 An image produced by slicing through a 3-D texture-mapped object

Medical imaging has traditionally been a key application for volume rendering because it allows the multiple 2-D images from a CT scanner or Magnetic Resonance Imaging system to be built into a three dimensional computer solid that can be sliced and probed to search for internal anomalies.

Seismic and petroleum applications also present 3-D volumetric data which can be sliced to probe for voids.

3-D texture mapping also solves some problems encountered at seams and joints with traditional 2-D texture mapping techniques. Because 3-D texture maps represent a consistent internal structure, rather than a 2-D image which is "stretched" over the surface of an object, they may provide a more realistic rendering for some kinds of materials.

For example, a piece of lumber rendered with 2-D texture mapping would show an external surface that resembled wood (grain, knots, etc.) but the internal portion of the rendered geometry would remain hollow. By using 3-D texture mapping, the same rendered geometry would allow slicing to reveal internal wood grain and other details.

Advanced Texture Mapping Techniques

A number of texture mapping techniques differ significantly enough from basic 2-D and 3-D texture mapping to warrant specific mention. Many of the techniques that follow are typically used to represent very high quality photorealistic images and are therefore not appropriate for interactive "real-time" texture mapping applications where a high level of interactivity is essential.

Procedural Mapping

2-D and 3-D texture mapping techniques differ from their more complex counterpart, *procedural mapping*, in that they make few assumptions about the object, and rely instead on a pre-existing (or pre-computed) texture to provide information regarding the appearance of the rendered object.

Procedural models describe objects that can interact with external events to modify themselves and can save space since they procedurally specify the objects to be drawn rather than storing lists of vertices. A procedural model of a sphere might take as a parameter the number of polygons to be used to render the sphere, and might allow a choice of functions to represent surface texture.

The burden of the representation of the rendered object lies with the logic of the procedure code. In simple terms, the procedure is customized to produce a specific appearance, usually placing emphasis on realism versus speed in rendering.

Procedural mapping does offer many benefits, including great flexibility in grasping the essence of objects without the constraint of following the laws of physics in representing the object's geometry (rendering a furry donut, for example). Further, procedural mapping allows the object to have varying degrees of quality in terms of physical appearance, based upon the user's control. Procedural modeling's drawbacks are specifically in the area of real-time rendering.

If procedural texture mapping is employed, procedural definitions of the color variations to be applied to the objects in a scene are used. This approach avoids the necessity of transformation calculations associated with mapping two-dimensional texture patterns onto object surfaces. However, the computation of these textures can require substantial compute resources which generally makes procedural texture mapping inappropriate for real-time texturing.

Procedural texture modeling and mapping is generally reserved for animation or high-quality photorealistic rendering where the computation can be performed in a batch manner and where the quality of the resulting image(s) is paramount.

Bump Mapping

Bump mapping is another technique which can be used to create interesting surface detail for realistic images employed in animation.

While texture mapping can be used to produce very fine surface detail, it suffers in attempting to model surface roughness. Thus, texture mapping is not useful for representing the appearance of fur, grass, or the details on the surface of an orange. Surface roughness is limited in that the details of illumination do not adequately correspond to the actual direct illumination in a scene.

Bump mapping applies a perturbation function to the surface normals of the object, and then uses those perturbed normals in the calculation of illumination of the object.

Displacement Mapping

A slight variation of bump mapping known as *displacement mapping* uses the textures themselves to move the surface, in addition to changing the surface's normals. The result of displacement mapping is the same as traditional bump mapping, with the addition of displacement appearance on the silhouette of the object.

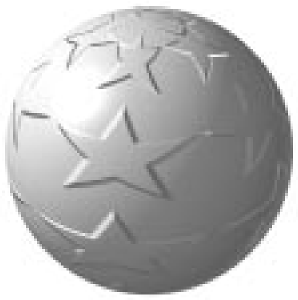

Figure 2-3 A displacement mapped sphere

Reflection Mapping

Reflection mapping (or environment mapping) can be used to provide highly realistic images of reflective objects placed in a three dimensional scene. Reflection mapping is supported by APIs like OpenGL and is applicable to interactive graphics rendering.

With reflection mapping, the ultimate goal is to simulate reflections from mirror-like or otherwise highly reflective surfaces. In that sense, reflection mapping may be thought of as a simplified ray tracing method. Essentially, a reflection mapping procedure computes the direction of reflection of a light ray from the viewer to the point being shaded. The texture is then scaled and translated to map onto the object based upon the viewer's point, the surface receiving the reflection, and the object being reflected.

Texture Mapping in Practice

Texture space is measured by the coordinates *u*, *v*, and *w*. The coordinate *u* represents horizontal direction, *v* represents vertical, and *w* represents depth (in the case of 3-D textures). This discussion will consider 2-D textures for simplicity.

As illustrated in Figure 2-4, the origin (*u*=0, *v*=0) of a texture is always located at the lower left corner. Both *u*, and *v*, range between 0 and 1, regardless of the size and shape of the texture.

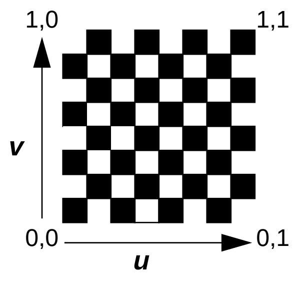

Figure 2-4 A texture in texture space

Texture coordinates differ from geometry coordinates in what is being measured. Geometry coordinates (*x*, *y*, and *z*), are a reference system for placing objects in a three dimensional world. Unlike a texture, a geometric object exists at a given set of coordinates in three-space. If a geometric object is transformed (moved, scaled, rotated, etc.), it exists at a different set of coordinates in three-space.

Repeating and Clamping Textures

Texture coordinates only measure texture repetitions. The upper right corner of the texture remains at (1,1), no matter how the texture is stretched or scaled. The space between (0,0), and (1,1) represents one repetition of the texture. If more than one repetition of the texture is used, the texture is said to be *tiled* (repeated) or *clamped*.

Repeating Textures (Tiling)

When textures are repeated, each tile adds 1 to the *u* and/or *v* coordinates such that a set of texture coordinates from 0,0, to 2,2 would be a grid of four tiles of the texture as shown in Figure 2-5.

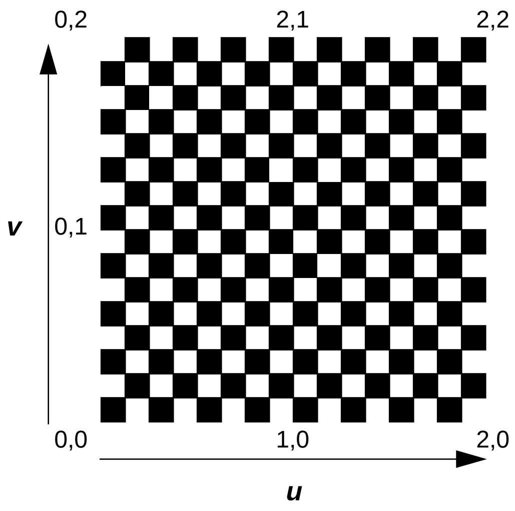

Figure 2-5 A tiled texture

Clamping Textures

The alternative to tiled textures is to specify texture clamping. With clamped textures, any texture values greater than 1 are "clamped" to 1 and any values less than 0 are clamped to 0. This technique is useful for applications that wish to provide a single copy of the texture on a large surface. Figure 2-6 illustrates the sample texture drawn as a clamped texture.

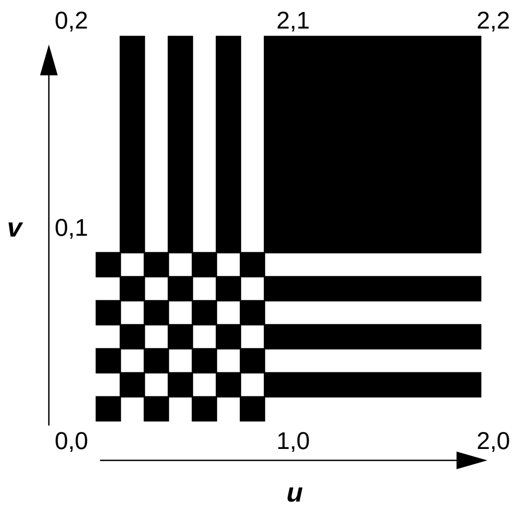

Figure 2-6 A clamped texture

Tiling and clamping can also be combined. For example, a texture could be clamped in the *u* direction, and tiled in the *v* direction.

Surface Space

Texture space is flat (or cubic in the case of 3-D textures). However, the surfaces that textures are placed on are often not flat at all. Thus, another set of coordinates is required to describe surface space.

Surface space is defined with the coordinates of *s* and *t*, with *s* representing the horizontal component, and *t* representing the vertical. *s* and *t* coordinates are equivalent to *u* and *v* coordinates but represent a different concept. Since the surface to be mapped may be very irregular, *s* and *t* coordinates provide a way of mapping a particular point on the texture (*u*,*v* coordinates) regardless of any stretching or distortion that may have taken place in order to place the 2-D texture onto the 3-D surface. In other words, the *u*,*v* coordinates of the 2-D image are referenced by the *s*, *t* coordinates on the surface that relate to the *x*, *y*, and *z* coordinates of the 3-D geometry.

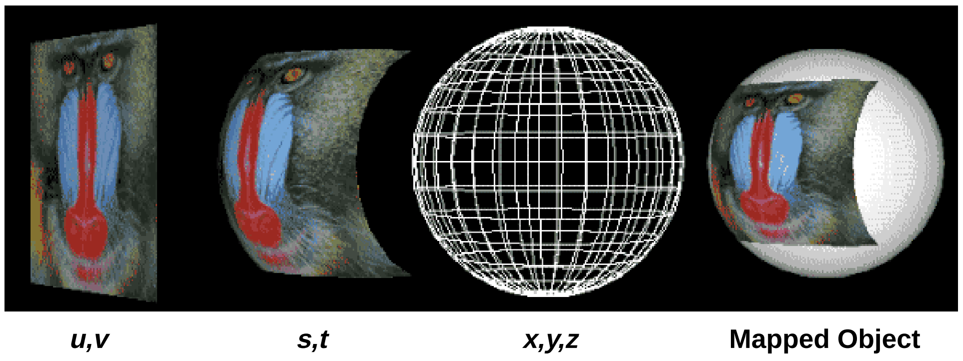

Figure 2-7 Different coordinate spaces required for texture mapping

Figure 2-7 illustrates the three different coordinate spaces used to map a 2-D texture onto a 3-D surface.

Mapping Textures to Surfaces

The simplest definition of texture mapping is the mapping of a defined texture to a specified surface. The geometric representation of the object and its perspective relative to the user's viewpoint complicate texture mapping because the texture must be transformed in order to cover the surface. This process is referred to as *parameterization*.

A 2-D texture is tied to its corresponding 3-D model by way of tie points which represent the mapping between texture coordinates, and coordinates on the object's surface. Mathematical projections are then employed to establish the correspondence between the two. The tie points can be provided by the program author (explicit), or some APIs provide facilities to automatically generate texture coordinates. For example, OpenGL provides for explicit, linear, and spherical projection of 2-D textures onto geometric objects.

The first step in automatic coordinate generation is to perform a projection of the object surface onto a *projection object*, such as a plane, cylinder, or sphere. The projection object is then unfolded onto a flat 2-D plane. The resulting 2-D surface corresponds to the texture map for the object. Figure 2-8 illustrates the projection of a geometrical object onto a projection surface.

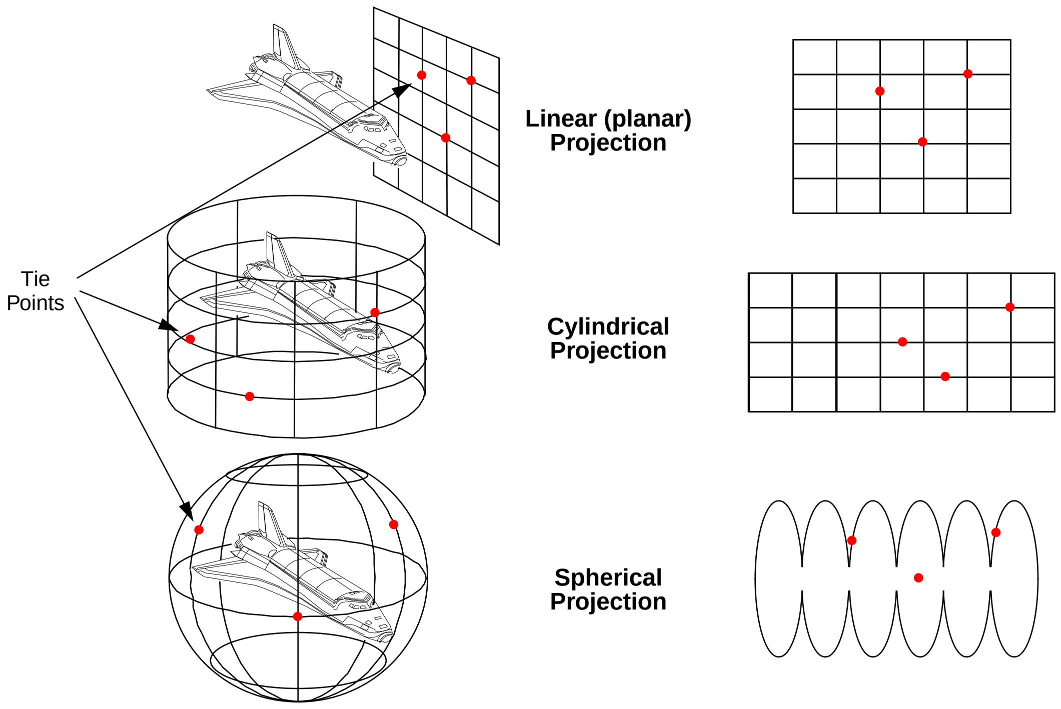

Figure 2-8 Texture mapping first requires that the object's geometry be projected onto a surface

Generally speaking, the plane and spherical projection methods are supported by interactive graphics APIs (OpenGL, XGL, PEXlib, etc.). Spherical projection is often used for reflection or environment mapping.

The projection process requires that the surface normals be used in finding the intersection of the object to the projection surface. The projection surfaces described above use either object centroid, normal, or linear projection methods, as illustrated in Figure 2-9.

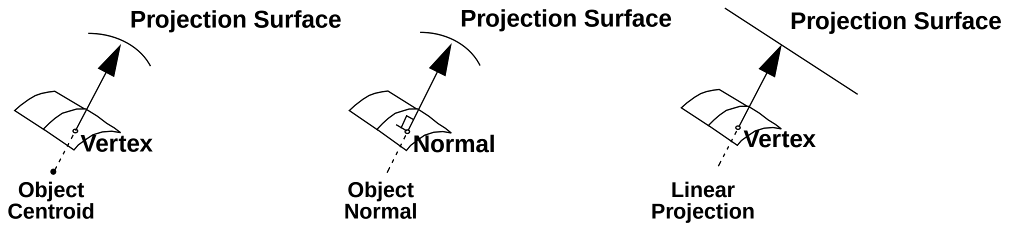

Figure 2-9 Alternative projection methods for texture mapping

Figure 2-10 illustrates a checkerboard pattern projected onto a spherical object using a variety of projection techniques. In general, the projection technique that most resembles the shape of the object will result in the most predictable result. Not all projection techniques are represented in all APIs.

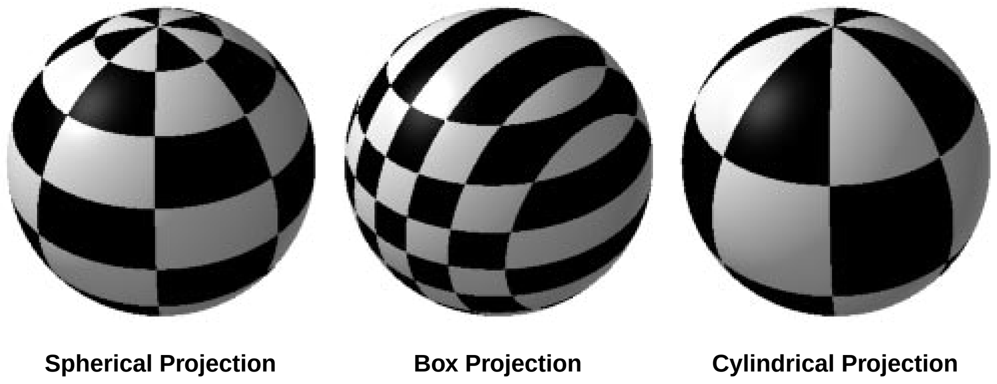

Figure 2-10 Several projections of a checkerboard pattern onto a spherical object

Texture Rendering Techniques

Additional techniques are needed to provide high visual-quality textures mapped to an endless variety of geometric objects. For example, a 128x128 texel image would be expected to appear correctly on a polygon that takes up a 128x128 pixel space on the workstation screen. However, if the projected polygon takes up twice the space of the texture, a method must be found to scale the image. Simply replicating pixels in the texture image results in a poor, blocky image.

Single Level Texture Filtering

To avoid low-quality texture-mapped images, texturing algorithms interpolate values from texels to arrive at the correct pixel color displayed on the screen. A variety of texture filtering techniques are used.

- *Single Sample*

  With a single sample approach, one sample, or value is taken from the texture map and used to set the screen pixel color. Single sample is the simplest filtering method and may produce artifacts in the resulting image.

- *Bilinear Interpolation*

  Bilinear Interpolation takes four samples to correctly approximate each displayed pixel. Bilinear interpolation provides a more visually correct image than single sampling, but less than trilinear interpolation.

- *Trilinear Interpolation*

  Trilinear interpolation represents the most computationally intensive filtering process but produces the most visually correct texture mapped image. Trilinear interpolation takes eight samples from the texture map to compute each screen pixel.

MIP Mapping

Though ideal for many situations, single level filtering techniques are not appropriate for all applications. Single level filtering techniques provide excellent results for pattern mapping onto polygons but require computing a filtered value at each point, so that for each pixel in the finished image, a filtering computation is performed. This approach results in a computationally expensive process where a single pixel in the final image may correspond to thousands of pixels in the source pattern (envision a textured, horizontal plane vanishing to infinity at the horizon).

MIP mapping (standing for *multum in parvo* — many things in a small place) addresses this problem by providing an approximation to an ideal filtering of an image. In MIP mapping, a series of filters is applied to the original texture image. Each successive filter results in an image half the size of the layer below it. Each of these filtered images constitutes a layer of the MIP map pyramid as shown in Figure 2-11. When a target pixel is covered by a collection of source pixels, the MIP map pixels corresponding to this collection most closely are used to give a filtered value.

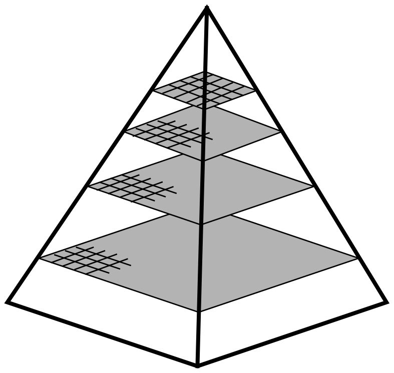

Figure 2-11 Texel pyramid employed in approximating colors for the texture map

In the illustration above, each pixel in a level *n* is equal to the average of four pixels on the level beneath. As a result, the resolution of each texture map is half of the level below, up to the highest level. At the top-most level, a single pixel represents the average color of the entire original base-level texture map.

The MIP mapping technique provides this advantage while only requiring a third again the amount of space required to store the original texture. Figure 2-12 illustrates a top-down view of the MIP map pyramid. The red, green, and blue channels of the original image fill three quarters of the MIP map. Each channel is then filtered by a factor of 4, and the R, G, and B components of the resulting image fill up three quarters of the remaining quarter. The process is continued until the MIP map is filled.

In general all of the techniques used for single sample filtering are available for generating the MIP map pyramid (single sample, bilinear interpolation, trilinear interpolation). Because the MIP map pyramid represents a hierarchy of filters applied to a single source image, the visual quality of a particular texture is strongly related to the characteristics of the filters used.

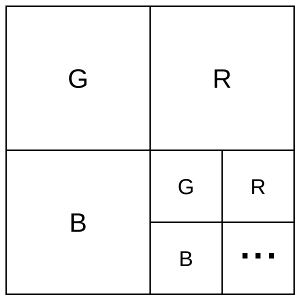

Figure 2-12 A top-down view of the MIP map pyramid

Texture Functions

Thus far textures have been described as being painted directly on the on the surface of an object. In actuality, multiple textures can be applied to a single surface and a variety of texture functions can be employed to determine how textures and surface colors are combined and masked.

The results of the texture functions depend on the format of the data stored in the texture map.

- *Decal*

  The decal function can be used to apply an opaque label to an object. If the texture data contains an alpha value, then the decal function can be used to apply an alpha blended texture, like a partially transparent decal, to the object.

- *Replace*

  The replace function simply replaces the surface color of the object with the colors represented in the texture.

- *Modulate*

  For modulation, the surface color is modulated by the contents of the texture map. Depending on the format of the texture data, the surface color is multiplied by values in the texture map.

  Modulation is particularly useful for use with lighting since the surface color can be used to attenuate the texture color.

- *Blend*

  The blend function can be used to blend the colors in the texture map with the existing colors on the surface. In some cases, an environment color can be used for blending as well.

Texture Mapping Architectures 3

Today's advanced graphics architectures must provide performance and high visual quality, and at the same time must represent a platform that is affordable and flexible enough to effectively support a wide range of applications. Satisfying these goals requires careful consideration of both hardware and software design issues.

Graphics hardware design places various stages of the 3-D graphics pipeline into dedicated graphics accelerators. Software design uses the hardware features available on a given platform and complements them by providing complete functionality coverage for a given API.

This section discusses different systems architectures for accelerating 3-D graphics applications and texture mapping in particular. Traditional approaches are contrasted with Sun's innovative Creator Graphics system architecture. Chapter 4 discusses software APIs which accelerate texture mapping.

Overview

Accelerating 3-D graphics primitives to provide interactive levels of performance is a large computational task. Today this computational problem is often addressed with hardware accelerators built from dedicated processors, memory, and ASIC technology.

Graphics commands enter the front of the 3-D graphics pipeline, and correctly transformed, shaded, and lit pixels are sent out the other end of the pipeline to the screen. A typical 3-D graphics pipeline is illustrated in Figure 3-1.

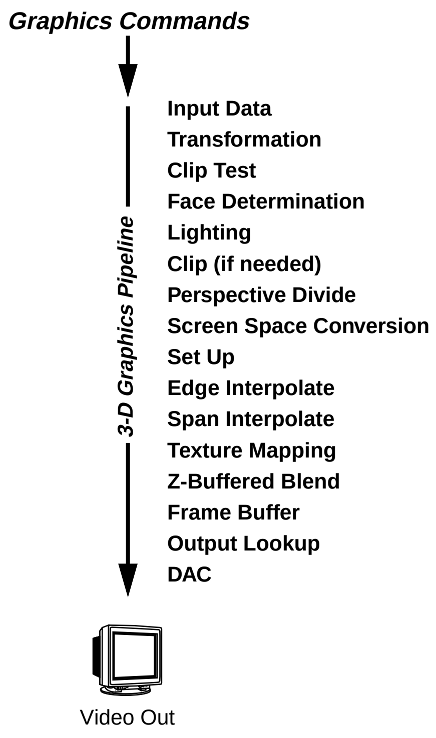

Figure 3-1 A typical 3-D graphics pipeline

Implementing 3-D pipelines in hardware can provide increased 3-D application performance. Unfortunately, 3-D hardware pipelines can present an all or nothing proposition. If an application requires a feature which is not implemented in hardware, the entire rendering process may need to be relegated to a much slower software pipeline. Given the interactivity requirements of today's complex, 3-D graphics applications, a slow software pipeline can render an application unusable.

Traditional Approaches to Hardware Texture Mapping

Accelerating texture mapping in the context of a pipelined graphics architecture requires additional resources above those typically needed for 3-D graphics. Because texture mapping implies a potentially different color for each rendered pixel, it must be performed within the 3-D pipeline as shown in Figure 3-1. Additionally, good texture mapping performance depends on fast access to potentially large images from the 3-D pipeline.

Unfortunately, pipelined 3-D graphics accelerators are often closed off from resources like the system processor and memory. As a result, most vendors with pipelined 3-D graphics accelerators have added dedicated texture storage memory in order to accelerate texture mapping. While this approach can produce the best performance for direct display and manipulation of simple texture mapped data, it does present some serious drawbacks for professional applications.

Limited Texture Capacity

Dedicated texture memory represents a finite resource and places distinct limits on the size and/or numbers of textures that can be represented. If a texture or textures won't fit in the dedicated texture memory, the system must resort to using a software pipeline for rendering 3-D graphics. Because texture mapping is most interesting for complex 3-D applications, this generally results in serious application performance degradation.

Texture Size/Memory Size	1MB	4MB	16MB	64MB
128x128 pixels	15	63	255	1023
128x128 pixels, MIP mapped	11	43	191	767
256x256 pixels	3	15	63	255
256x256 pixels, MIP mapped	2	11	43	191
512x512 pixels	0	3	15	63
512x512 pixels, MIP mapped	0	2	11	43
1024x1024 pixels	0	0	3	15
1024x1024 pixels, MIP mapped	0	0	2	11
Largest Single Texture size	256x256	512x512	1024x1024	2048x2048

Table 3-1 An example of memory sizes and supported texture capacities

The numbers in Table 3-1 indicate how many textures of a given size can be fit into 1, 4, 16, and 64 MB of texture memory. Numbers are given for both simple textures and MIP-mapped representations.

It is clear that for sufficiently large or numerous textures, applications will eventually fail to fit in the available dedicated memory resources and will, therefore, fall back to using a slower software process.

Limited Application Deployment

Applications like seismic analysis and medical imaging require large 3-D textures. These applications are often precluded from running with adequate performance on architectures with dedicated, but finite, texture storage.

Limited Texture Access

Dedicated texture storage memory is necessarily located on the graphics accelerator, remote from the system processor and memory. Because of this relative distance, invalidating and reloading the texture content can cause significant delays. These delays can easily neutralize the performance gains ordinarily obtained for smaller texture maps.

Increased System Cost

As shown in Table 3-1, a considerable amount of dedicated texture memory is required in order to support textures of a useful size. This dedicated, special-purpose memory increases the cost of the system and represents a wasted resource when not in use.

Sun's Creator Graphics System

In contrast, Sun's Creator graphics systems present an alternative to dedicated texture storage without resorting to the slow speeds incurred with an unaccelerated software pipeline.

Sun's Ultra Creator3D graphics systems feature a highly integrated approach that connects powerful UltraSPARC™ processors with advanced Creator3D accelerator modules over a high-speed packet-switched memory interconnect known as the Ultra Port Architecture (UPA).

This innovative approach allows textures to be stored in general purpose system memory where they are operated on by powerful multimedia instructions in the UltraSPARC CPU. Known as the VIS™ Instruction Set, these instructions provide key mathematical and multimedia instructions, many of which are useful for texture mapping operations.

This approach has many advantages for providing texture mapping support:

- *Low System Cost*

  Leveraging system memory for texture storage keeps system costs low since system memory is a shared resource. When not in use for texture mapping applications, memory remains available for other applications.

- *Large Texture Storage*

  Extremely large textures can be accommodated in system memory since it represents a virtual address space. Applications like 3-D texture mapping which use very large textures can be easily supported.

- *Accelerated Pixel Processing (Rasterization)*

  The VIS Instruction Set provides instructions which allow the UltraSPARC processor to directly access and operate on image (pixel) data with a high degree of parallelism. Other SPARC and VIS instructions provide facilities for formatting and moving data at very high rates of speed across the UPA memory interconnect to the frame buffer.

- *High Precision for Texture Mapping Operations*

  Texture mapping is a part of a pipelined process. The results of one operation serve as the input for another operation, with only the final image being displayed to the screen. With Creator Graphics systems, these intermediate pixel colors are stored as 16-bit or 32-bit fixed-data values in fast system memory. This added precision and dynamic range is useful for filtering and image computations on pixel values.

- *Texture Caching and Swapping*

  By performing texture mapping in the CPU, and storing textures in main memory, the system cache and Memory Management Unit (MMU) can be used to considerable advantage.

  Since many texture mapping functions, like interpolation, operate on neighboring pixels, the system cache can provide fast access to image data.

The MMU can be utilized to access large images held in system memory (which are often striped and can exceed the cache) allowing them to be operated on directly by the CPU in the same address space as other application data.

- *Scalable Performance*

  Because texture mapping is performed in the UltraSPARC processor, it can benefit from the scalability gained by adding faster UltraSPARC processors.

- *Other Interesting Texture Mapping Applications*

  By locating textures in main memory, other interesting texture mapping applications are available, such as performing image processing functions on textures and using texture mapped video rather than static images.

This flexible architecture enables Sun to add dedicated texture storage in future generations of the Creator Graphics family without loosing the advantages of the existing architecture. The Creator3D system, its architecture, and specific approach to texture mapping are contained in the sections that follow.

Creator3D System Architecture

The Creator3D architecture available in Sun's Ultra workstation product line represents a unified system designed to accelerate 2-D and 3-D graphics, imaging, and video applications. Graphics was central to the design of the Ultra systems from the beginning, enabling engineers to build on lessons learned with other architectures, allowing them to locate graphics technology where it would most benefit performance.

This approach resulted in a highly-integrated, modular architecture that tightly couples the CPU, the system memory interconnect, and the frame buffer and graphics accelerator technology. In Creator Graphics systems, graphics processing is balanced across the entire system to take advantage of all available resources. Table 3-2 lists the different parts of the system responsible for accelerating different graphics operations.

Functionality	Responsible System Component
Window System and 2-D Graphics	Creator Graphics Module
Imaging, Video, and Texture Mapping	UltraSPARC (VIS Instruction Set) Creator Graphics Module
3-D Graphics Pipeline	UltraSPARC (Floating Point Unit), Creator Graphics Module

Table 3-2 Components responsible for processing in the Creator Graphics system

The Creator Graphics module accelerates the window system and 2-D graphics applications as well as providing color space conversion for video decompression and playback. The UltraSPARC processor and the VIS instruction set perform most imaging, texture mapping, and video decompression tasks. The UltraSPARC processor and the Creator Graphics subsystem together share the task of accelerating the 3-D graphics pipeline.

3-D Graphics Rendering Pipeline

On Creator3D systems, the 3-D graphics pipeline is handled by *both* the UltraSPARC CPU and the Creator Graphics module. Figure 3-2 compares the Creator3D graphics pipeline with that of earlier ZX graphics systems from Sun.

With accelerators like the ZX, it was common to use dedicated floating point processors (e.g., ZXFloat) to compute the front-end portion of the 3-D graphics pipeline. In Creator Graphics systems, the considerable floating point performance of the UltraSPARC processor provides this functionality.

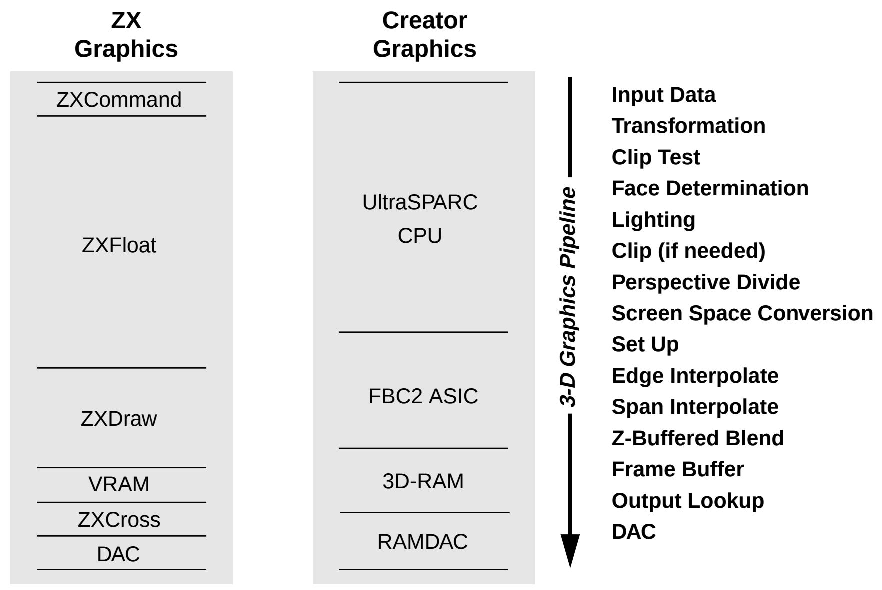

Figure 3-2 Pipeline differences for SPARCstation 20ZX and Creator Graphics systems

As the figure illustrates, the front portion of the 3-D graphics rendering pipeline contains such floating point intensive operations as transformations, clip tests, face determination, lighting, perspective divide, and conversion to screen space coordinates.

Only with recent advances in processor technology have general purpose processors been available with the necessary floating point processing power to implement a design like Creator Graphics. By using fast general purpose processors like UltraSPARC, it has finally become possible to build inexpensive systems that provide high-end 3-D performance.

Figure 3-3 illustrates the high level Creator Graphics system architecture and identifies which components perform various activities in the 3-D rendering pipeline.

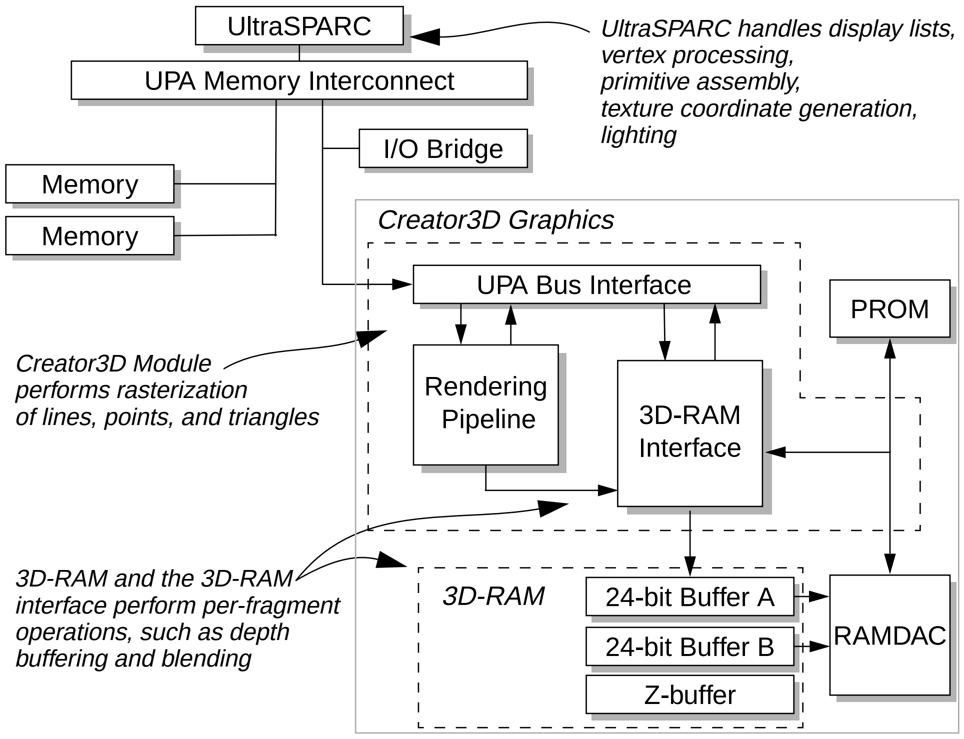
UltraSPARC
UPA Memory Interconnect
UltraSPARC handles display lists, vertex processing, primitive assembly, texture coordinate generation, lighting
I/O Bridge
Memory
Memory
Creator3D Graphics
UPA Bus Interface
PROM
Creator3D Module performs rasterization of lines, points, and triangles
Rendering Pipeline
3D-RAM Interface
3D-RAM and the 3D-RAM interface perform per-fragment operations, such as depth buffering and blending
3D-RAM
24-bit Buffer A
24-bit Buffer B
Z-buffer
RAMDAC

Figure 3-3 Creator Graphics high-level block diagram

Rasterization and Pixel Processing Architecture

As shown in Figure 3-4, rasterization and pixel processing is performed on the Creator3D graphics system in one of three ways depending on the primitives being rendered

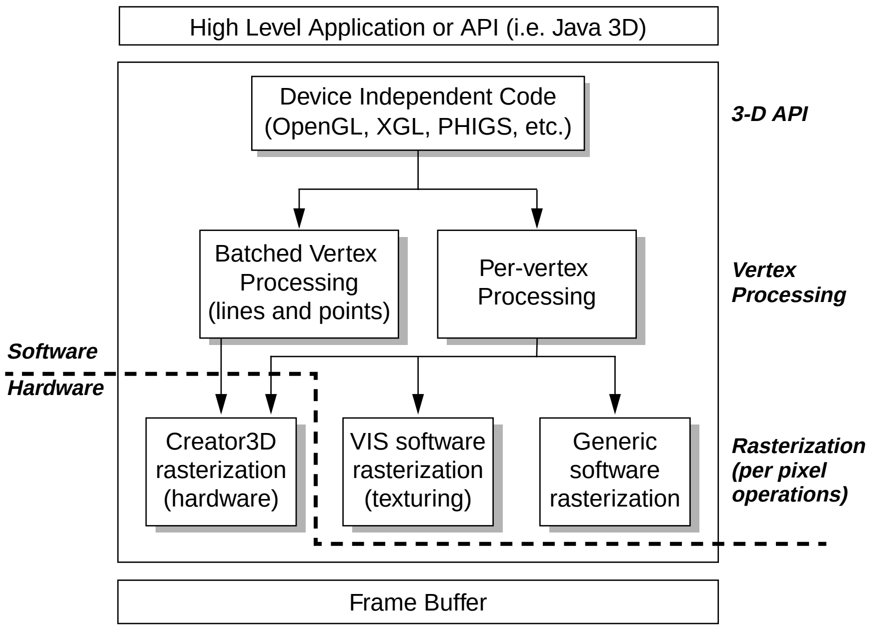
High Level Application or API (i.e. Java 3D)
Device Independent Code (OpenGL, XGL, PHIGS, etc.)
3-D API
Batched Vertex Processing (lines and points)
Per-vertex Processing
Vertex Processing
Software
Hardware
Creator3D rasterization (hardware)
VIS software rasterization (texturing)
Generic software rasterization
Rasterization (per pixel operations)
Frame Buffer

Figure 3-4 Creator3D features three separate rasterization paths

- *Creator3D Hardware Rasterizer*

  The Creator3D Graphics subsystem handles lines, points, and triangles, and does simple pixel processing such as blending and the depth-buffer test. Rasterization is done by the Frame Buffer Controller 2 ASIC (FBC2), and pixel processing is performed by 3D-RAM.

- *The VIS Software Rasterizer*

  Texture rasterization is done by the UltraSPARC processor using optimized code that employs the VIS instruction set.

- *Generic Software Rasterizer*

  Most APIs also provide a software rasterizer which uses the UltraSPARC processor to do rasterization in an a less optimized fashion.

Table 3-3 summarizes the available data paths through the Creator3D graphics system.

Vertex Processing	Rasterization	Performance
Batched Vertex Processor	Creator3D Graphics Rasterizer	Fastest Path
Per-vertex Processor	Creator3D Graphics Rasterizer	Fast Path
Per-vertex Processor	VIS Rasterizer (Texture Mapping)	Optimized for Texture Mapping
Per-vertex Processor	Generic Software Rasterizer (if available with API)	Slow Path

Table 3-3 Data paths through the Creator3D graphics subsystem

For more information on the Creator Graphics architecture, refer to the *Creator Graphics Technical White Paper.*

VIS™ Instruction Set

UltraSPARC was the first microprocessor to fully support advanced graphics and multimedia data manipulation. By introducing a comprehensive set of multimedia instructions, known as the VIS Instruction Set, UltraSPARC provides extremely fast hardware support for 2-D and 3-D graphics, video and audio processing, and image manipulation. VIS instructions are grouped into the following areas:

- *Pixel format and conversion (expand, pack, merge)*
- *Image Processing (partitioned add, subtract, multiply, array addressing)*
- *Real-time video compression (motion estimation)*
- *Data transfer and animation speed-up (64-bit block load/store)*

The graphics unit in UltraSPARC relies on the integer registers for addressing image data and the floating point registers for manipulating image data. This division of duty between the integer and floating point registers enables UltraSPARC to make use of all available internal registers, maximizing throughput.

Pixel information in UltraSPARC can be represented as eight 8-bit, four 16-bit, or two 32-bit integer values. These values can be used to represent the color (RGB) and intensity information for a color image. For higher resolution images, like those used in medical or color imaging, UltraSPARC can use 16-bit components. Support is provided for both band-interleaved images, with the various color components stored together, and band-sequential images that have all of the values for one color component stored together.

Intermediate results for advanced image manipulation are stored as 16- or 32-bit, fixed-data values. These provide an intermediate format with enough precision and dynamic range for filtering and image computations on pixel values. UltraSPARC has several single-cycle instructions specifically tailored for manipulating these 16- and 32-bit components.

UltraSPARC also includes a variety of instructions that are essential for advanced image manipulation. For example, UltraSPARC supports a filtering operation for scaling, rotating, and smoothing images. The filtering operation processes four pixels at a time, giving UltraSPARC an *order of magnitude* performance advantage over other processors.

VIS Texture Mapping Implementation Details

The application of pixel-level texture mapping potentially produces a different color per pixel in the resulting image. As a result, the triangle scan interpolation algorithm in the 3-D pipeline is replicated for texture mapping utilizing VIS instructions.

For every pixel in a given triangle, the location is calculated, the color is looked up from the texture according to the calculated *u,v* value, then interpolation, texturing operations, and lighting are applied to determine the final pixel color. Finally, the pixel position and color is sent down to the Creator3D frame buffer.

These steps make up the VIS-enabled software rasterizer mentioned previously. A generic VIS-enabled rasterizer is illustrated in Figure 3-5. Note that the VIS-enabled rasterizer differs depending on the 3-D graphics API that it implements. Not all APIs provide the same functionality or the same order of processing.

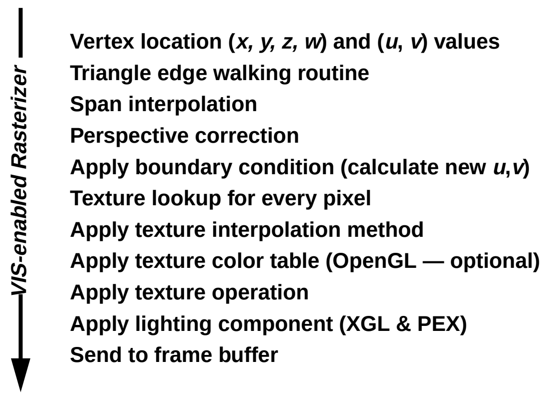
VIS-enabled Rasterizer
Vertex location (x, y, z, w) and (u, v) values
Triangle edge walking routine
Span interpolation
Perspective correction
Apply boundary condition (calculate new u,v)
Texture lookup for every pixel
Apply texture interpolation method
Apply texture color table (OpenGL — optional)
Apply texture operation
Apply lighting component (XGL & PEX)
Send to frame buffer

Figure 3-5 Pipeline describing a generic VIS-enabled software rasterizer

A number of VIS instructions are used to accelerate texture mapping operations within the VIS-enabled software rasterizer.

- *Accelerating Texture Interpolation Methods*

  VIS is very useful for computing the various sampling methods used in texture mapping (e.g. bilinear interpolation, MIP mapped bilinear, MIP mapped trilinear, and MIP map point linear interpolations).

  For example, in a bilinear case, 4 sample colors need to be extracted from the neighboring texels and weight-averaged together. Using common "C-style" coding techniques, a red, green, blue, and alpha value would need to be extracted from each of the four colors, then each would need to be multiplied, added, and shifted four times.

  Using VIS, the four operations (for red, green, blue, and alpha) are performed simultaneously using VIS floating point add, multiplication, and pack instructions. Again, intermediate results are stored as 16-bit values for greater precision.

- *Accelerating Texture Operations*

  VIS instructions can be effectively used to accelerate the calculation of texturing operations (modulate, blend, decal, etc.).

  For example, the modulate texture operation calculation requires that the red, green, and blue values for the pixel be separated, each multiplied with the object color (r, g, and b), and the result packed together.

  By using VIS instructions, this operation is performed for the r, g, and b values in parallel for each pixel using a single 8x16-bit floating point multiply instruction and a single floating point pack instruction.

- *Applying Lighting Components*

  Lighting calculations utilize parallel 8x16 bit floating point multiplication, 16-bit floating point add, and 16-bit pack instructions to operate on all of the color components of a pixel in parallel. Intermediate results are stored as a 16-bit floating point value for added precision.

- *Moving Pixels to the Frame Buffer*

  Because the VIS-enabled rasterizer is running in the UltraSPARC processor, an in-line SPARC function (*write-double*) is used to write the Z value and the color to the pixel address in the frame buffer.

Software Interfaces for Texture Mapping 4

All applications which perform texture mapping require a graphics application programming interface to handle the complex operations and the close interaction required with platform hardware. Developers may have a variety of issues which drive their selection of a graphics interface, including performance, functionality, cross-platform considerations, and underlying hardware support.

The following sections briefly describe the use of key graphics software interfaces for texture mapping on Sun. OpenGL, XGL™, and PEXlib, are briefly discussed. For more information on these software interfaces, please reference their respective product documentation and specifications.

Texture Mapping with OpenGL

The OpenGL graphics application programming interface is a vendor neutral software interface which operates independently of operating and window system platforms. Based upon its proprietary predecessor, GL, OpenGL provides a broad set of 3-D graphics functions for modeling transformations, color, lighting, shading, and advanced features such as texture mapping.

The OpenGL Architecture Review Board is responsible for defining OpenGL's characteristics and features, as well as conformance testing, release approval, and specification definition. OpenGL is independent of any underlying window system, and display of rendered graphics occurs through either extensions to the graphics library or through direct window system calls.

Solaris native OpenGL functionality is available now from Sun as an unbundled product. First introduced for Creator3D graphics on the Ultra 1 and 2 systems, *Solaris OpenGL 1.1* now provides a highly accelerated implementation of OpenGL for the entire range of Creator and Creator3D systems and an optimized software implementation for many other graphics accelerators.

Native OpenGL is fully integrated with Solaris, allowing developers to take advantage of the Solaris operating environment's advanced features. Solaris OpenGL can run with Common Desktop Environments (CDE) or OpenWindows™ environments. A defined common extension to the X Window System allows OpenGL client to run across distributed heterogeneous networks.

The OpenGL architecture/pipeline is illustrated in Figure 2-5.

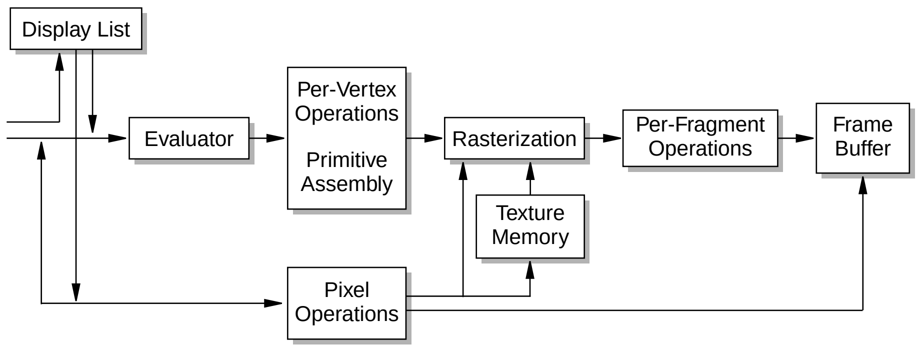

Figure 4-1 The OpenGL architecture and pipeline

Solaris™ OpenGL software represents a high performance implementation of OpenGL on the Creator Graphics platform. Specific optimizations include:

- Accelerating OpenGL by using all of the features of the Creator3D graphics subsystem
- Optimizing line and point transformation and clip test, and some subsets of texture lookup and filtering
- Implementing OpenGL to its complete specification by writing code for primitive assembly and vertex processing including coordinate transformations, texture coordinate generation, and clipping

- Providing two forms of software rasterization for OpenGL for features which are not rasterized in the Creator Graphics hardware rasterizer: a rasterizer which utilizes the UltraSPARC VIS Instruction Set for texturing functions, and a generic software rasterizer for all features not handled by the hardware or by VIS.

OpenGL 3-D Texture Mapping Extension

An extension to OpenGL now exists for performing 3-D texture mapping. Both in-memory formats for 3-D images and pixel storage modes are provided.

Solaris OpenGL 1.1 supports and accelerates the OpenGL 3-D texture mapping extension.

For more information on OpenGL, refer to *The Design of the OpenGL Interface* written by Mark Segal and Kurt Akeley. For the complete OpenGL specification see *The OpenGL Graphics System: A Specification* written by the same authors.

Texture Mapping with the XGL™ Graphics Library

XGL is a software library of 2-D and 3-D graphics primitive functions designed to support a wide variety of graphics-based applications. The XGL library provides immediate mode, non display-list functionality.

The XGL library implicitly uses hardware graphics acceleration whenever possible, and most of its functionality is independent of the underlying hardware. Hardware graphics acceleration occurs automatically without explicitly requesting it from within an XGL program. In cases such as hardware double buffering, the functionality is device dependent, and the XGL library does not simulate the functionality in software.

The XGL library is designed to insulate the application programmer from the specifics of the window system and the underlying hardware device with XGL *objects*, which describe virtual components of the graphics rendering system. XGL objects simplify the work of the application programmer by presenting a consistent, device-independent graphics model. For example, display devices, such as X11 windows, are known to the XGL programmer as Window Raster Device objects. Graphics state information describing how XGL graphics primitives are drawn on the device is stored in XGL 2-D Context or 3-D Context objects.

2-D Texture Mapping with the XGL Graphics Library

In XGL, the 2-D texture to be draped on the geometric primitive can be stored as an array of colors that will eventually be mapped onto the polygonal surface. The surface to be textured is specified with vertex coordinates and texture coordinates (*u*,*v*), the latter being used to map the color array on the polygon's surface. The *u* and *v* are interpolated across the span and then used as indices into the texture map to obtain the texture color. This color is combined with the primitive color (obtained by interpolating vertex colors across spans) or the colors specified by the application to obtain a final color value at the pixel location.

Texturing is achieved in XGL through a set of flexible attributes. XGL also allows the result of the texturing operation to be applied to different stages of the rendering pipeline (see Figure 4-2), and supports sampling methods such as point, bilinear, and trilinear to obtain texture value and color composition (e.g., blend, decal, and modulate). Finally, XGL additionally has the capability of applying multiple textures to individual primitives.

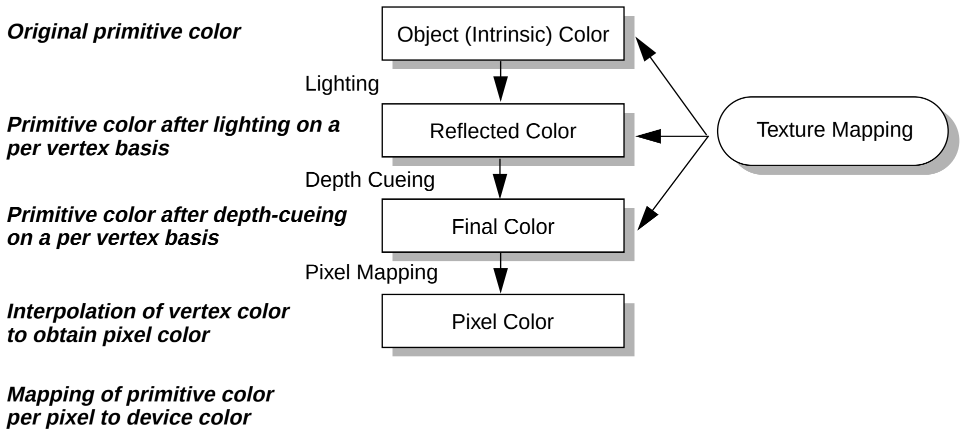
Original primitive color
Object (Intrinsic) Color
Lighting
Primitive color after lighting on a per vertex basis
Reflected Color
Texture Mapping
Depth Cueing
Primitive color after depth-cueing on a per vertex basis
Final Color
Pixel Mapping
Interpolation of vertex color to obtain pixel color
Pixel Color
Mapping of primitive color per pixel to device color

Figure 4-2 XGL rendering pipeline

Texture Mapping with PEXlib

PEXlib is available through Sun as an unbundled product, meeting current industry standards complying with the PEXlib 5.2 specification. PEXlib is optimized for Sun platforms to take full advantage of available graphics processor capabilities including transformations, lighting, and shading, advanced primitive support, and texture mapping.

PEXlib is an industry standard application program interface designed to perform 3-D rendering both locally and across X Windows desktops. PEXlib has significant origins in both PHIGS and PEX.

PEX was developed as an extension to the X Windows X protocol, providing the ability to communicate 3-D floating point coordinates, and supporting common PHIGS functionality and methodology.

PEXlib users usually find PEXlib similar to PHIGS, but with slightly richer functionality, direct mode rendering (vs. display list based), and many other features. PEXlib is a cross-platform graphics API. It is supported by virtually every UNIX® workstation vendor, and is part of several major standards definitions including those from X/Open and the Open Software Foundation.

Like Sun's XGL Graphics Library, PEXlib supports MIP mapping, texture compositing, and tiling capabilities. Additional texturing technology such as environment mapping is also part of PEXlib. Bump mapping is not supported.

The texture mapping functions available with PEXlib include:

- PEXCreateMipmapTM
- PEXCreateMipmapTMFromResources
- PEXFreeTextureMap

Texture mapping utilities built in to PEXlib include:

- PEXFreeMipmap
- PEXGenerateMipMap
- PEXGenerateMipMapFromResources
- PEXOCCTMCoordFillAreaSet
- PEXOCCTMCoordIndexedFillAreaSets
- PEXOCCTMCoordIndexedTriangles
- PEXOCCTMCoordPolyTriangle

- PEXOCCTMCoordQuadrilateralMesh
- PEXOCCTMCoordTriangleFan
- PEXOCCTMCoordTriangles
- PEXOCCTMCoordTriangleStrip

For additional discussion of PEXlib programming, consult the *PEXlib Programming Manual*, from O'Reilly and Associates.

3-D Graphics API Texture Mapping Comparison

Table 4-1 provides a functionality comparison among XGL, PEXlib, and OpenGL. For more information, refer to the documentation specific to the API.

Feature	XGL	PEXlib	OpenGL
Client/Server Protocol Support	PEX	PEX	Proprietary
Color Domain — 1 Dimensional Texture Maps	■	■	■
Color Domain — 2 Dimensional Texture Maps	■	■	■
Coordinate Source — *t0* (1 Dimension)	■	■	■
Coordinate Source — *t1* (2 Dimension)	■	■	■
Coordinate Source — *t2* (3 Dimension)	■	■	■
Coordinate Source — *t3* (4 Dimension)		■	■
Coordinate Source — *Variable* Dimensions		■	
Capture MIP maps from User Data	■	■	■
Capture MIP maps from X Windows	■	■	■
Default Texture Map (checkerboard)			
Front/Backfaces	■		■
Minification Filters - Single Base	■	■	■
Minification Filters - Linear Base	■	■	■
Minification Filters - Single Base in MIP Map	■	■	■
Minification Filters - Linear Base in MIP Map	■	■	■
Minification Filters - Single Base between MIP Map	■	■	■
Minification Filters - Linear Base between MIP Map	■	■	■
Magnification Filters - Single Base	■	■	■
Magnification Filters - Single Base	■	■	■
Transformation Matrix Orientation (Lookup Table Equivalence)	■	■	■
Hints for Texture Dimensions (Acceleration)	■		■
Hints for Maximum Number of Active Textures	■		■
Paramterization Method - Explicit	■	■	■
Paramterization Method - Linear, World Coordinates	■	■	■

Feature	XGL	PEXlib	OpenGL
Paramterization Method - Linear, Virtual Coordinates	■	■	■
Paramterization Method - Reflect Sphere, Virtual Coordinates	■	■	■
Paramterization Method - Reflect Sphere, World Coordinates	■	■	■
Pespective Correction - None			■
Perspective Correction - Pixel	■	■	■
Texture Maps -2^n Size Required	■	■	■
Texture Maps - Pre-Lighting Rendering Order	■	■	■
Texture Maps - Post-Lighting Rendering Order	■	■	
Optimization Hints Available	■		■
Square Texture Map Required	No	No	No
Surface Primitives - Fill	■	■	■
Surface Primitives - Fill of Set of Areas	■		■
Surface Primitives - Triangle Strip	■	■	■
Surface Primitives - Quadrilateral Mesh	■	■	■
Compositing - Replace			■
Compositing - Modulate	■	■	■
Compositing - Decal	■	■	■
Texture Pixel (Texel) Type Support - RGB	■	■	■
Texture Pixel (Texel) Type Support - RGBA	■	■	■
Texture Pixel (Texel) Type Support - Luminance	■	■	■
Texture Pixel (Texel) Type Support - Luminance Alpha	■	■	■
Texture Pixel (Texel) Type Support - Intensity			■
Texture Pixel (Texel) Type Support - Alpha			■
Boundary Conditions - Clamp Color, Absolute Source	■	■	■
Boundary Conditions - Clamp Color, Explicit Source	■	■	■
Boundary Conditions - Boundary			■
Boundary Conditions - Boundary with Wrap	■	■	■
Boundary Conditions - Boundary with Mirroring	■	■	■
Boundary Conditions - Boundary with Border Color	■	■	■

Table 4-1 Texture mapping support feature comparison

Summary 5

In recent years, advanced graphics technologies such as texture mapping have moved from specialized niche applications to general purpose usage in broad consumer-oriented markets. Entertainment, animation, simulation, and even graphic arts have benefited from the low cost of texture-capable desktop systems. Together with advances in texture mapping algorithms and low-cost 3-D graphic subsystems and intelligent, highly integrated, memory controller devices, the sophistication and general purpose application of texturing technology is enabling increased rendering realism across virtually all segments of graphics use.

Sun Microsystems™ is the largest provider of graphics systems for virtually every market segment. Long heralded as a leader in platform and graphics device integration, Sun and its third parties provide the industry's broadest family of solutions. Regardless of what level of performance or cost a customer requires, these solutions enable the effective use of texture mapping technology, freeing the user to focus on results rather than on the intricacies of the application.

Particularly exciting is the use of texture mapping in extremely price-sensitive environments where the cost per seat is paramount. Sun's highly integrated low-cost Ultra workstations with Creator3D graphics systems enable users to apply texturing to their data with a minimal impact on their budget.

Sun Microsystems' open systems philosophy and wide support for third party products and industry standards such as OpenGL and PEXlib further broaden the choices for end-users, thus enabling them to implement cross-platform capable applications without compromising portability or performance.

References A

OpenGL Programming Guide, Second Edition, Addison Wesley Developer's Press.

Solaris XGL Reference Manual, SunSoft Inc.
Solaris XGL Programmer's Manual, SunSoft Inc.

SunPHIGS Reference Manual, Sun Microsystems Computer Company.
SunPHIGS Extensions Reference Manual, Sun Microsystems Computer Company.
SunPHIGS Porting Guide, Sun Microsystems Computer Company.

OpenWindows User's Guide, SunSoft Inc.
OpenWindows Programmers Guide, SunSoft Inc.

X Window System User's Guide, O'Reilly & Associates. Inc.
Xlib Reference Manual, O'Reilly & Associates. Inc.
Xlib Programming Manual, O'Reilly & Associates. Inc.

X Protocol Reference Manual, O'Reilly & Associates. Inc
PEX Protocol Specification Version 5.0P, X Consortium.

PEXlib Specification and C Language Binding, Hewlett-Packard Company, Version 5.2, In Deevlopment,

PEXlib Programming Manual, O'Reilly & Associates. Inc., 1992.

An Introduction to Computer Graphics Concepts, Addison-Wesley, 1991.

The Renderman Companion, Steve Upstill, Addison-Wesley, 1990.

Survey of Texture Mapping, P. Heckbert, IEEE Computer Graphics and Applications, Vol. 6, No. 11, 1986.

Image Synthesis, Theory and Practice, D. Thalman and N. Magnenat Thalman, Springer-Verlag, 1987.

Glossary B

Abstract Coordinate Space (AC)

An abstract coordinate space in which the geometrical object is defined. Every geometrical object (or portion of the object) can be described in this space, together with descriptors of all information which describes the object's color.

CLS

Color space. A colorspace may be RGB, HIV, CMYK, or RGBA. Individuals elements of a color space are known as colors.

Composition

A function of one or more variables applied to the texture map.

Device Coordinate Space (DC)

Device coordinate space refers to the two-dimensional coordinates of the computer display device. Device coordinates for computer graphics displays are typically represented by a set of 2-D coordinate pairs corresponding to the integer pixel values of the display.

Geometrical Object

A two or three-dimensional shape, constructed from discrete coordinates. In a broader context, a geometrical object can also be a pixel, span, scanline, primitive, image, or object.

Interpolation

A way of calculating a value that falls between other, known values. In image processing, interpolation frequently plays a part in geometric operations such as rotation. After that type of spatial transformation, pixel locations in the output image will correspond to non-integer coordinates in the input image. Therefore, the pixel values in the output must be calculated by looking at the values of the pixels surrounding the point of interest in the input.

Lookup Table

Lookup tables are used for general image modification. Each entry in a lookup table contains an index—a value that may appear in the source image—and a value or set of values to be written to the destination image. For each pixel in the source, a table-lookup function finds the pixel's value on the index side of the table and then writes the output value or values for that entry to the corresponding pixel in the destination.

MIP Map

A technique which improves the visual quality of a texture map through precomputation (filtering) of texture map pixel values. This approach delivers significant performance and quality value for textures that are applied repeatedly within an individual scene. MIP maps are based upon pre-computed area sampling designed to reduce the computation and storage required to approximate an average color for a large number of pixels.

Model Coordinate Space (MC)

The coordinate space in which the coordinates of an object or related objects are represented. Objects in model coordinates must be transformed to world and device coordinates prior to rendering on a computer graphics display.

Nearest Neighbor Interpolation

This method takes the value it is looking for to be the value of the pixel closest to the point of interest. This type of interpolation is the fastest type, but can introduce artifacts in the output image; for example, smooth lines in the input image may show up as jagged lines in the output. Nearest neighbor interpolation is sometimes called zero-order interpolation.

Parameterization

For each transformation of a texture map, a parameterization may be applied between the input coordinates and the output coordinates.

Texture Coordinate Space (TC)

Simplistically, texture coordinate space refers to the coordinate space of a texture map, and typically consists of 2-D coordinates *u,v*.

Texture Map Filtering

An algorithm that transforms the texture map to add or eliminate effects. Such effects may include blurring, aliasing and antialiasing, rippling, ringing, MIP mapping, resampling, etc.

World Coordinate Space (WC)

The overall coordinate space in which all elements of a scene, including objects and light sources are represented prior to transformation to device coordinates.

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
901 San Antonio Road
Palo Alto, CA 94303 USA
415 960-1300
FAX 415 969-9131
http://www.sun.com

Sales Offices

Argentina: +54-1-311-0700
Australia: +61-2-9844-5000
Austria: +43-1-60563-0
Belgium: +32-2-716-7911
Brazil: +55-11-524-8988
Canada: +905-477-6745
Chile: +56-2-638-6364
Colombia: +571-622-1717
Commonwealth of Independent States:
+7-095-956-5470
Czech/Slovak Republics:
+42-2-205-102-33
Denmark: +45-44-89-49-89
Estonia: +372-6-308-900
Finland: +358-0-525-561
France: +33-01-30-67-50-00
Germany: +49-89-46008-0
Greece: +30-1-680-6676
Hong Kong: +852-2802-4188
Hungary: +36-1-202-4415
Iceland: +354-563-3010
India: +91-80-559-9595
Ireland: +353-1-8055-666
Israel: +972-9-956-9250
Italy: +39-39-60551
Japan: +81-3-5717-5000
Korea: +822-3469-0114
Latin America/Caribbean:
+1-415-688-9464
Latvia: +371-755-11-33
Lithuania: +370-729-8468
Luxembourg: +352-491-1331
Malaysia: +603-264-9988
Mexico: +52-5-258-6100
Netherlands: +31-33-450-1234
New Zealand: +64-4-499-2344
Norway: +47-2218-5800
People's Republic of China:
Beijing: +86-10-6849-2828
Chengdu: +86-28-678-0121
Guangzhou: +86-20-8777-9913
Shanghai: +86-21-6247-4068
Poland: +48-22-658-4535
Portugal: +351-1-412-7710
Singapore: +65-224-3388
South Africa: +2711-805-4305
Spain: +34-1-596-9900
Sweden: +46-8-623-90-00
Switzerland: +41-1-825-7111
Taiwan: +886-2-514-0567
Thailand: +662-636-1555
Turkey: +90-212-236-3300
United Arab Emirates:
+971-4-366-333
United Kingdom: +44-1-276-20444
United States: +1-800-821-4643
Venezuela: +58-2-286-1044
Worldwide Headquarters:
+1-415-960-1300

Printed in USA